COFlood: Concurrent Opportunistic Flooding in Asynchronous Duty Cycle Networks

Abstract— **In energy constraint wireless networks, radios usually operate in duty cycle mode. With low maintenance cost, asynchronous duty cycle radio management is widely adopted. To achieve fast network flooding is challenging in asynchronous duty cycle networks. Recently, concurrent flooding is a promising approach to the performance of network flooding. In concurrent flooding, the key challenge is how to select a set of concurrent senders to improve both flooding speed and energy efficiency. We observe that selecting neither large nor small number of concurrent senders can achieve the optimal performance in different deployments. In this paper, we propose COFlood (Concurrent Opportunistic Flooding), a practical and effective concurrent flooding protocol in asynchronous duty cycle networks. The basic idea is based on an energy-efficient flooding tree, COFlood opportunistically select extra concurrent senders that can speed up network flooding. First, COFlood constructs an energy-efficient flooding tree in distributed manner. The non-leaf nodes are selected as senders and they can cover the entire network with low energy consumption. Moreover, we find that exploiting both early wake-up nodes and long lossy links can speed up the flooding tree based network flooding. Then, COFlood develops a light-weight method to select the nodes that meet the conditions of these two opportunities as opportunistic senders. We implement COFlood in TinyOS and evaluate it on two real testbeds. In comparison with state-of-the-art concurrent flooding protocol, completion time and energy consumption can be reduced by up to 35.3% and 26.6%.**

I. Introduction

In many IoT applications [11] [12], nodes are usually unattended and energy constraint. To extend network lifetime, asynchronous duty cycle radio management, namely LPL (Low Power Listening) (i.e. Box-MAC [13], Zisense [20]) is widely adopted. In asynchronous duty cycle networks, all nodes periodically turn on their radios, but follow different active schedules. In order to achieve reliable broadcast, a sender has to continuously transmit the same packet for a whole sleep interval. Hopefully, a receiver can obtain a packet at the rendezvous.

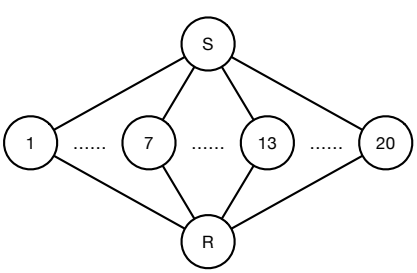
Network flooding is a fundamental approach to propagate message to all devices in several system services, such as time synchronization [6], system parameters update [18] and binary image dissemination [21]. In network flooding, a set of nodes (called *senders*) cover all nodes through multi-hop relay. We use *completion time* to depict the flooding speed. The completion time is the duration between the sink starts to broadcast and the last node successfully receives the flooding packet. The completion time should be as short as possible. However, in asynchronous duty cycle networks, the long channel occupation of a LPL broadcast increases the probability of channel contention and packet collision. Both channel contention and packet collision can lead a node may miss some early chances of packet receiving. Due to the long sleep interval, the receiving time of the node can be greatly delayed after missing a early chance. As a result, the completion time is further enlarged.

To alleviate the negative influence of channel contention and packet collision, *Chase* [1] utilizes capture effect to develop concurrent flooding. With *Chase*, all nodes can immediately broadcast the received flooding packet without any backoff. In this way, a node cannot miss any early chance of packet receiving. We empirically study the influence of concurrent sender selection on the completion time and energy efficiency (Section II). We find that completion time is not optimal when all nodes are selected as concurrent senders. The energy is also overused. Moreover, we observe that the completion time and energy efficiency are still not optimal when the number of concurrent senders is too small. Hence, the question here is *how to adaptively select a set of concurrent senders to achieve the shortest completion time and keep the energy consumption as low as possible*.

In this paper, we propose COFlood (Concurrent Opportunistic Flooding), a practical and effective concurrent flooding protocol in asynchronous duty cycle networks. To achieve the shortest completion time and keep the energy consumption low, COFlood consists of two parts. First, COFlood constructs an energy-efficient flooding tree. The non-leaf nodes are selected as senders that cover the entire network with low energy consumption. We show that it is a NP-hard problem to construct the energy-optimal flooding tree. COFlood develops a heuristic algorithm to construct the flooding in distributed manner. Second, based on the flooding tree, we find two opportunities, namely early wake-up nodes and long lossy links, can speed up concurrent flooding. COFlood develops a light-weight method to enable a node captures these two opportunities. The nodes are selected as opportunistic senders when they meet these two opportunities.

We implement COFlood in TinyOS and evaluate its performance on two real testbeds. The results show that COFlood outperforms *Chase* on the two testbeds in terms of both completion time and energy efficiency. The main contributions are listed as follows:

- We empirically study the performance of state-of-the-art concurrent flooding and identify the sender selection is the crucial challenge to enable efficient concurrent flooding in asynchronous duty cycle networks.
- We propose COFlood, a practical and efficient concurrent flooding protocol with a novel sender selection method,



(a) Control Topology

(b) Lab Testbed

Fig. 1: Illustration of the control experiment topology and Lab testbed environment.

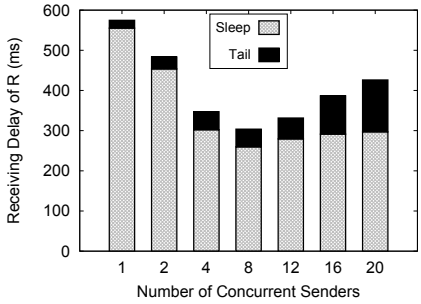


Fig. 2: The reception delay of **R** with different number of sender on control topology.

to achieve both short completion time and low energy consumption.

- We evaluate COFlood on two real testbeds. In comparison with *Chase*, the completion time can be reduced by up to 35.3%. The energy consumption can be reduced by up to 26.6%.

The rest paper is organized as follows. Section II carefully studies the performance of state-of-the-art concurrent flooding. The detailed design of COFlood protocol is shown in Section III. We show the implementation details and evaluation results in Section IV and V, respectively. The related work is introduced in Section VI. Finally, we conclude our work in Section VII.

II. Empirical Study

Chase proposed concurrent flooding to speed up network flooding. *Chase* selects all nodes as concurrent senders. We doubt that involving all nodes are not a wise choice. Next, we study the detailed influence of sender selection in concurrent flooding.

A. Influence of Sender Selection

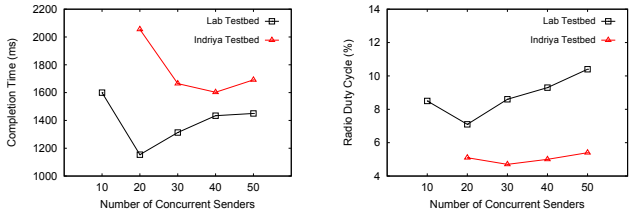
1) Control Topology: First, we use a two-hop topology, which contains one sink (**S**), twenty hop-1 nodes ({**1, 2, ..., 20**}) and one hop-2 node (**R**) as shown in Figure 1(a). {**1, 2, ..., 20**} can reliably communicate with each other, **S** and **R**. For all nodes, the sleep interval is set as 512ms. In comparison with selecting all 20 hop-1 nodes, we randomly select 1, 2, 4, 8, 12, 16 and 20 hop-1 nodes as concurrent senders. We use *Chase* with the default settings [1]. For each setting, **S** continuously initializes network flooding 100 times. We evaluate the receiving delay of **R**, which is the duration from **S** starts to broadcast to **R** receives a flooding packet. The receiving delay of **R** consists of two parts. One part is *sleep time*, which is the duration from **S** starts to broadcast to **R** wakes up and detects the concurrent broadcast of {**1, 2, ..., 20**}. The other part is *tail time*, which is the time **R** successfully receives a flooding packet after it wakes up.

The results are shown in Figure 2. We can see when the number of concurrent senders increases from 1 to 8, the sleep time is reduce from 555.2ms to 259.6ms. If a sender can successfully receive a flooding packet before the earliest wake-up of **R** after **S** starts to broadcast, the sender is a *early wake-up node*. The early wake-up nodes provides the chances that **R** can receive a flooding packet once it wakes up so that the receiving delay of **R** is short. With the increasing of the number of concurrent senders, the opportunities of early wake-up node increases, as a result, the receiving delay of **R** is getting shorter.

Moreover, although the sleep time is comparable when the number of concurrent senders increases from 8 to 20, the tail time increases from 44.1ms to 129.3ms. The reason is when the number of concurrent senders is large, **R** needs more time to meet the condition of capture effect so that successfully receive a flooding packet. As a result, the receiving delay increases from 303.7ms to 426.3ms.

2) Real Testbed: We further conduct experiments on two real testbeds. One (called *Lab Testbed*) consists of 50 TelosB nodes on a wall of our laboratory as shown in Figure 1(b). The other is Indriya Testbed [3], which has 56 TelosB nodes. We set the radio channel as 26 and 22 for Lab Testbed and Indriya Testbed. Correspondingly, we set the transmission power as 2 and 31. The average number of hop-1 neighbors (i.e., packet reception ratio between two nodes is higher than 0.8) is 10 and 6. The network diameter is 4 and 6 hops. The sleep interval is set as 512ms. We select 50, 40, 30, 20 and 10 nodes as concurrent senders. Then, we evaluate *Chase* with the default settings [1] in terms of completion time and average radio duty cycle of all nodes. For each setting, the sink continuously initializes network flooding 100 times. The packet payload is set as 40 bytes.

The results are shown in Figure 3. Since 10 nodes cannot fully cover the entire network on Indriya Testbed, we get rid of this setting. We can see the curves of completion time and radio duty cycle are V-shape on both testbeds. The completion time is minimum when the number of concurrent senders is 20 and 40 on Lab Testbed and Indriya Testbed. Moreover, selecting 20 and 30 concurrent senders can achieve the minimum radio duty cycle on Lab Testbed and Indriya Testbed. When we select more concurrent senders, the longer tail time increases the completion time. More concurrent senders also consume more energy. When we select less concurrent senders, the longer sleep time increase the completion time. Moreover,



(a) Completion Time (b) Energy Consumption

Fig. 3: The average completion time and radio duty cycle of concurrent flooding by selecting different number of concurrent senders on Lab Testbed and Indriya Testbed.

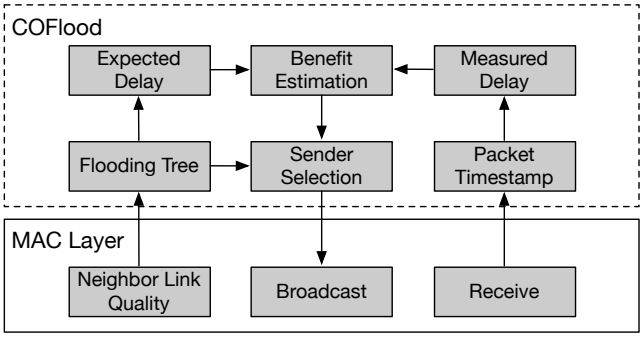


Fig. 4: System overview of COFlood.

due to the possible packet loss, rebroadcast is needed to ensure the full coverage. Hence, the radio duty cycle becomes higher as well.

B. Practical Hints

According to our empirical study, selecting neither all nodes nor a small set of nodes as senders is a wise choice. To achieve efficient concurrent flooding, how to select concurrent senders is a critical and challenging problem. We summarize two practical hints for concurrent sender selection. One is the number of senders is better to be small to keep the tail time short and the energy consumption low. The other is some opportunities (e.g., early wake-up node) should be further exploit to speed up concurrent flooding.

III. COFlood Design

In this section, we illustrate the detailed design of COFlood. The objective of COFlood is to optimize the completion time, while keep the energy consumption as low as possible. The system overview of COFlood is shown in Figure 4. COFlood sits on MAC layer, which provides the Tx/Rx primitives of concurrent broadcast [1] and neighbor link quality. COFlood contains two parts to select concurrent senders. First, it constructs an energy-efficient flooding tree (Section III-A and Section III-B). Then, *sender selection* selects the non-leaf nodes as senders, which can cover the whole network with minimum energy cost. Moreover, COFlood exploits two opportunities, namely long lossy links and early wake-up nodes, to speed up concurrent flooding (Section III-C). Based on the flooding tree, each node calculates its *expected delay* to receive a flooding packet. With *packet timestamp*, COFlood measures the per-hop receiving delay as *measured delay*. Combining both expected delay and measured delay, *benefit estimation* determines whether a node meets the conditions of the two opportunities. Further, *sender selection* selects the leaf nodes with high benefits as senders.

The network model are abstracted as follow. For node i, the number of its neighbors is N_i. $\{n_i^1, n_i^2, ..., n_i^{N_i}\}$ indicates all i's neighbors. n_i^j is one of i's neighbors. The link from n_i^j to i is indicated as $l_i(n_i^j)$. The successful packet reception ratio (PRR) of link $l_i(n_i^j)$ is $p_i(n_i^j)$. The sleep interval is T.

A. Energy Efficient Flooding Tree

We illustrate the construction of the energy-efficient flooding tree, which selects a set of senders to cover the entire network with minimum energy cost. In concurrent network flooding, the total energy consumption E_f consists of three parts, which are signal sampling E_s, tail time E_t and broadcast cost E_b. Next, we give the detailed analysis of these energy costs, then define the formal optimization problem.

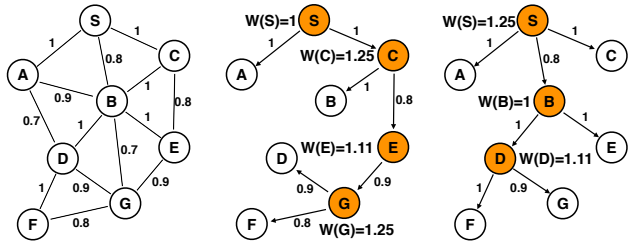
E_s is the baseline energy for detecting the possible concurrent broadcast in LPL. Since signal sampling is periodically triggered, E_s is constant during a fixed period. It is irrelevant to the number of selected concurrent senders. Moreover, E_t is the energy used to receive the flooding packet when concurrent broadcast is detected. According to Section II-A, E_t is related to the number of concurrent senders. The more the concurrent senders are, the longer the tail time is. To make E_t low, we should make the number of concurrent senders as small as possible.

E_b is the energy cost that all senders spend on broadcast. If a node is not a sender, its broadcast cost is zero. Otherwise, its broadcast cost is the number of broadcasts which can cover all its children on the flooding tree. For sender s_i, we use $W(s_i)$ to indicate its expected number of broadcasts. n_i indicates the number of its children. Given its children set $\{c_{s_i}^1, c_{s_i}^2, ..., c_{s_i}^{n_i}\}$ and the corresponding link PRR $\{p_{s_i}(c_{s_i}^1), p_{s_i}(c_{s_i}^2), ..., p_{s_i}(c_{s_i}^{n_i})\}$, $W(s_i)$ is determined by the neighbor with the worst link PRR. $W(s_i)$ can be calculate as follow:

$$W(s_i) = \mathbf{max}\{\frac{1}{p_{s_i}(c_{s_i}^j)} | \forall j \in [1, n_i]\} \quad (1)$$

Given a set of senders $\{s_1, s_2, ..., s_m\}$, the total energy cost E_b is $\sum_1^m W(s_i)$.

Taking Figure 5 as an example, Figure 5(a) illustrates the network topology. The number on an edge indicates the PRR of the link. Figure 5(b) shows an example of flooding tree. The nodes {S, C, E, G} are selected as senders to cover the whole network. For S, its children are {A, C}. Hence, W(S) is 1. Similarly, W(C), W(E) and W(G) are 1.25, 1.11 and 1.25. The corresponding E_b is 4.61. Moreover, Figure 5(c) shows another set of senders {S, B, D}. For S, its children becomes {A, B, C}. W(S) increases to 1.25 since $p_C(B)$ is 0.8. Similarly, W(B) and W(D) become 1 and 1.11. The



(a) Network Topology (b) Flooding Tree 1 (c) Flooding Tree 2

Fig. 5: Overview of the flooding tree structure.

corresponding E_b is 3.36. Although W(S increases, E_b is reduced since the {S, B, D} has smaller number of senders than {S, C, E, G}. Meanwhile, E_t is also reduced. If a set of senders can achieve the minimum E_b, it can usually keep E_t low as well since the total number of senders is bounded.

Overall, to achieve the minimum energy cost E_f approximately equals to obtain minimum E_b. However, the E_b minimization problem is *Minimum Wight Connected Dominating Set* problem which is NP-hard. Next, we develop a distributed heuristic algorithm of flooding tree construction to achieve an approximate solution.

B. Tree Sender Selection Algorithm

We define two metrics EBQ (Expected Broadcast Quality) and PEC (Path Energy Cost). For a node, if it acts as a sender and covers a set of children, its EBQ is the ratio between the expected number of broadcasts and the number of its children. It indicates the average energy cost spending on a child. In Figure 5(b), S covers 2 neighbors (A and C) and W(S) is 1, then its EBQ is 0.5. As shown in Figure 5(c), S covers 3 neighbors (A, B and C) and W(S) is 1.25, its EBQ is 0.42. The smaller the EBQ is, the more efficient the energy usage is. The larger the expected number of broadcasts is, the larger EBQ is. On the opposite, the larger the number of children is, the smaller EBQ is. Hence, if a node has small EBQ, either the quality of the links between it and its children is good or it covers a large set of children. For node i, we use EBQ(i) to represent its EBQ.

Moreover, PEC is the cumulative EBQ of the senders along the path from sink to it. For node i, we use PEC(i) to depict its PEC. In Figure 5(b), PEC(F) is calculated as follow:

$$\sum_{i \in \{\text{S, C, E, G}\}} \text{EBQ}(i) = 0.5 + 0.63 + 1.11 + 0.63 = 2.87 \quad (2)$$

In comparison, as shown in Figure 5(c), PEC(F) becomes smaller as follow:

$$\sum_{i \in \{\text{S,B,D}\}} \text{EBQ}(i) = 0.42 + 0.5 + 0.55 = 1.47 \quad (3)$$

The larger the PEC is, the more energy cost is accumulated along the path. If we greedily keep the PEC of each node as small as possible, the overall energy cost of concurrent flooding is small. For node i, $P(i)$ indicates its parent node on the flooding tree.

Algorithm 1 Node i Tree Sender Selection Algorithm

Input: N_i neighbors $\{n_i^1, n_i^2, ..., n_i^{N_i}\}$; the corresponding EBQ $\{\text{EBQ}(n_i^1), \text{EBQ}(n_i^2), ..., \text{EBQ}(n_i^{N_i})\}$, PEC $\{\text{PEC}(n_i^1), \text{PEC}(n_i^2), ..., \text{PEC}(n_i^{N_i})\}$ and parent node $\{P(n_i^1), P(n_i^2), ..., P(n_i^{N_i})\}$; new beacon from neighbor n_i^j, its current EBQ(n_i^j), PEC(n_i^j) and parent node $P(n_i^j)$.
Output: i is selected as sender or not, the number of broadcast $W(i)$.
1: Update the EBQ(n_i^j), PEC(n_i^j) and $P(n_i^j)$.
2: According to Algorithm 2, update local PEC and $P(i)$.
3: According to Algorithm 3, update local EBQ and $W(i)$.
4: **for** each k, $k \in [1, N_i]$. **do**
5: **if** $P(n_i^k)$ equals to i. **then**
6: i is selected as a sender.
7: **end if**
8: **end for**

Each node maintains its local EBQ, PEC and parent node. Moreover, it records the EBQ, PEC and parent node of its neighbors. Different nodes exchange their local information through broadcasting *tree beacons* (Section IV-B). Then, every node determines whether it is a non-leaf node (i.e., tree sender) in a distributed manner. For node i, the tree sender selection algorithm is shown in Algorithm 1. After i receive a beacon from a neighbor n_i^j, i extracts the current EBQ(n_i^j), PEC(n_i^j) and $P(n_i^j)$ from the beacon as input. First, i updates the information of the neighbor n_i^j (line 1). Then, i updates local PEC(i), $P(i)$, EBQ(i) and $W(i)$ according to Algorithm 3 and Algorithm 2 (line 2-3). If a neighbor selects i as its parent node, i is selected as a tree sender (line 4-8). Next, we describe the algorithms used to update EBQ, $W(i)$, PEC and $P(i)$.

Algorithm 2 illustrates the process that node i updates PEC(i). According to the definition of PEC, node i can recursively compute PEC(i) given its parent node $P(i)$ as follow:

$$\text{PEC}(i) = \text{PEC}(P(i)) + \text{EBQ}(P(i)) \quad (4)$$

Hence, according to the observed PEC and EBQ of all neighbors, the principle is to select the neighbor which provides the smallest PEC(i) as parent node.

Algorithm 2 takes the link quality, EBQ, PEC and number of broadcasts of all neighbors as input. If node i is sink, PEC(sink) and P(sink) are set as 0 and NULL (line 2). Otherwise, node i initializes its parent and PEC(i) as NULL and PEC-MAX, which indicates the theoretical maximum PEC (line 4). Then, it searches all neighbors (line 5). For a neighbor n_i^j with valid parent node $P(n_i^j)$, node i calculates the temporary PEC$^t(i)$ if n_i^j is selected as its parent, and the required number of broadcasts $W^r(n_i^j)$, which indicates how much broadcasts are needed if n_i^j covers i (line 6 and 7). If n_i^j can provide more energy-efficient coverage and meet the requirement of $W^r(n_i^j)$, node i updates PEC(i) and $P(i)$ as PEC$^t(i)$ and n_i^j (line 8-11).

Moreover, in Equation 4, EBQ is an important metric for parent selection. Algorithm 3 illustrates how node i updates EBQ(i). Node i has the potential to improve the PEC of its neighbors whose PEC is currently larger than PEC(i). We call

Algorithm 2 Node i PEC Update Algorithm

Input: N_i neighbors $\{n_i^1, n_i^2, ..., n_i^{N_i}\}$; the corresponding link quality $\{p_i(n_i^1)$, $p_i(n_i^2)$, ..., $p_i(n_i^{N_i})\}$, EBQ $\{$EBQ(n_i^1), EBQ(n_i^2), ..., EBQ$(n_i^{N_i})\}$, PEC $\{$PEC(n_i^1), PEC(n_i^2), ..., PEC$(n_i^{N_i})$ and expected number of broadcasts $\{W(n_i^1), W(n_i^2), ..., W(n_i^{N_i})\}\}$.
Output: Local PEC PEC(i), parent node $P(i)$.

1: **if** Node i is sink. **then**
2: Set PEC(sink) as 0 and P(sink) as sink.
3: **else**
4: Initialize PEC(i) as PEC-MAX, $P(i)$ as NULL.
5: **for** each j, $j \in [1, N_i]$ and $P(n_i^j)$ is not NULL. **do**
6: Temporary PEC$^t(i)$=PEC(n_i^j)+EBQ(n_i^j).
7: Required number of broadcasts $W^r(n_i^j) = 1/p_i(n_i^j)$.
8: **if** PEC$^t(i)$<PEC(i) and $W^r(n_i^j) \leq W(n_i^j)$ **then**
9: PEC(i)=PEC$^t(i)$
10: Set $P(i)$ as n_i^j.
11: **end if**
12: **end for**
13: **end if**

Algorithm 3 Node i EBQ Update Algorithm

Input: Local PEC PEC(i), N_i neighbors $\{n_i^1, n_i^2, ..., n_i^{N_i}\}$, the corresponding link quality $\{p_i(n_i^1)$, $p_i(n_i^2)$, ..., $p_i(n_i^{N_i})\}$, EBQ $\{$EBQ(n_i^1), EBQ(n_i^2), ..., EBQ$(n_i^{N_i})\}$ and PEC $\{$PEC(n_i^1), PEC(n_i^2), ..., PEC$(n_i^{N_i})\}$.
Output: Local EBQ EBQ(i), expected number of broadcasts $W(i)$.

1: According to link quality, sorting neighbors in descending order. The list of sorted neighbors is $\{\overline{n}_i^1, \overline{n}_i^2, ..., \overline{n}_i^{N_i}\}$
2: **if** $P(i)$ is NULL. **then**
3: Set EBQ(i) as EBQ-MAX and $W(i)$ as 0.
4: **else**
5: Initialize EBQ(i) as EBQ-MAX, the number of children n_i as 0.
6: **for** each j, $j \in [1, N_i]$. **do**
7: **if** PEC$(\overline{n}_i^j)$ < PEC(i) **then**
8: $n_i = n_i + 1$
9: Temporary $W^t(i) = 1/p_i(\overline{n}_i^j)$
10: Temporary EBQ$^t(i) = W^t(i)/n_i$
11: **if** EBQ$^t(i) \leq$ EBQ(i) **then**
12: $W(i) = W^t(i)$
13: EBQ(i) = EBQ$^t(i)$
14: **end if**
15: **end if**
16: **end for**
17: **end if**

these neighbors as *child candidates*. To update EBQ(i), the basic idea is to select a set of child candidates to achieve the smallest EBQ(i). In this way, node i can provide the most competitive PEC to attract its child candidates to select it as parent node.

Specifically, according to the link quality of all neighbors, node i sorts them in descending order indicated as $\{\overline{n}_i^1, \overline{n}_i^2, ..., \overline{n}_i^{N_i}\}$ (line 1). In this way, according to Equation 1, $W(i)$ equals to $1/p_i(\overline{n}_i^j)$ for covering the sub-set of neighbors $\{\overline{n}_i^1, ..., \overline{n}_i^j\}$. Then, if node i has no valid parent node, EBQ(i) and $W(i)$ are set as EBQ-MAX, which is the upper bound of EBQ, and 0 (line 3). Otherwise, EBQ(i) and the number of children n_i are initialized as EBQ-MAX and 0 (line 5). For each neighbor, if it is a child candidate, node i updates the number of children n_i (line 8), then calculates the temporary $W^t(i)$ and EBQ$^t(i)$ (line 9 and 10). If EBQ$^t(i)$ has better broadcast quality, node i correspondingly updates EBQ(i) and $W(i)$ (line 11-14).

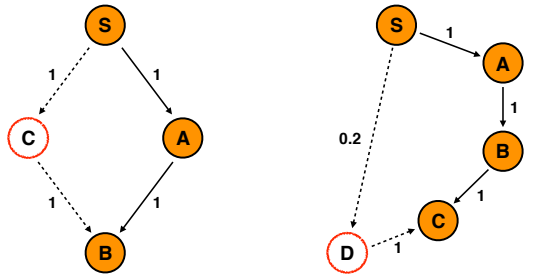
Let us take Figure 5(a) as an example to show how the algorithms work. At the beginning, according to Algorithm 2, sink S sets its parent node and PEC(S) as S and 0. S can cover A, B and C, whose PEC is PEC-MAX yet. According to Algorithm 3, it updates EBQ(S) and W(S) as 0.42 and 1.25. After A, B and C receive a beacon from S, they updates their PEC as 0.42, their parent node as S. Now, since A can only cover D, both EBQ(A) and W(A) are 1.43 and 1.43. Similarly, C can only cover E. EBQ(C) and W(C) are 1.25 and 1.25. In contrast, since B can cover D, E and G, EBQ(B) and W(B) are 0.48 and 1.43. After D receives the beacons from A and B, it will choose B as its parent node since PEC(D) is 0.95 smaller rather than selecting A. Similarly, E will choose B, but not C. G selects B as its parent node as well. PEC(D), PEC(E) and PEC(G) are 0.9. Since E have no neighbor whose PEC is smaller than PEC(E), it has no potential to cover any nodes. EBQ(E) and W(E) is EBQ-MAX and 0. Moreover, since both D and G can cover F, EBQ(D) and EBQ(G) is 1 and 1.25. Since D can provide smaller PEC(F) (1.9) than G (2.15), F will select D as its parent node. Finally, according to Algorithm 1, after several rounds of beacons, S, B and D know that they are selected as senders. Their broadcast cost W(S), W(B) and W(D) are 1.25, 1.43 and 1, respectively. In comparison with the flooding tree shown in Figure 5(c), the total broadcast energy cost E_b is 3.68, which is a little higher. Since the set of senders is the same, the tail energy cost E_t is similar. Thus, our distributed sender selection algorithm can achieve approximate optimal set of senders.

C. Fast Flooding Opportunities

As we have observed in Section II, if the number of concurrent senders are small, although the energy efficiency is kept, it will lose some early wake-up nodes so that the completion time of network flooding may be large. Here, based on the energy-efficient flooding tree (Section III-A), we define and exploit two potential opportunities to speed up network flooding.

As shown in Figure 6. The first opportunity is to take the advantage of the early wake-up nodes, who can shorten the sleep delay. In Figure 6(a), A is the parent node of B, and its parent node is sink S. C firmly connects with S and B. After S initializes a network flooding, if A wakes up later than B, B has to wait for another sleep interval. However, if C wakes up earlier than B, selecting C as a temporary sender can shorten the sleep delay of B. The early wake-up nodes has been observed through the control experiments in Section II-A. This opportunity is called *shortcut path*.

The other opportunity is to exploit long lossy link to shorten the sleep delay. In Figure 6(b), C is three hops away from sink S. A and B are the two senders along the path from S to C. D connects with C and the link between S and D is a lossy link. Since the link PRR is only 0.2, it is too low to be utilized by flooding tree. However, if D has successfully received a flooding packet from S over the lossy link, it can cover C with two-hop delay, which is shorter than the three-hop path on flooding tree. This opportunity is called *long link*.



(a) Opportunity 1 (b) Opportunity 2

Fig. 6: Two kinds of early wake-up opportunities, C in (a) and E in (b), to speed up network flooding on flooding tree.

To exploit these two opportunities, the challenge is how can a node recognize these opportunities with local information. We develop two metrics ETD (Expected Tree Delay) and MPD (Measured Per-hop Delay) to capture these two opportunities. ETD indicates the expected sleep delay on flooding tree. For node i, its ETD is indicated as ETD(i). The active schedules of all nodes comply with uniform distribution [10]. For node i, the expected per-hop sleep time is $T/2$. If i does not successfully receive the packet, the sleep time increases an extra sleep interval T. Given the link quality $p_i(P(i))$ (indicated as p_i for short) between i and its parent node $P(i)$, the expected per-hop sleep time t_i is calculated as follow:

$$\begin{aligned} t_i &= p_i\frac{T}{2} + ... + (1-p_i)^n p_i(nT + \frac{T}{2}) + ... \\ &= p_i\frac{T}{2}\sum_{i=0}^{\infty}(1-p_i)^i + p_i T\sum_{i=1}^{\infty} i(1-p_i)^i \\ &= \frac{T}{2} + T\frac{1-p_i}{p_i} = \frac{T}{p_i} - \frac{T}{2} \end{aligned} \tag{5}$$

Given the parent node $P(i)$, i can calculate ETD(i) as follow:

$$\text{ETD}(i) = \text{ETD}(P(i)) + t_i \tag{6}$$

We set ETD(sink) as 0. Then, every node embeds its ETD into its beacon. Hence, after i receives a new tree beacon from $P(i)$, it can recursively calculate its ETD.

Moreover, for node i, its MPD(i) indicates the actual per-hop delay between a sender starts to broadcast and the time i successfully receives the flooding packet. To enable i can obtain MPD(i), we attach the MAC layer timestamp [18] on each preamble packet of *Chase* broadcast. As it shown in Figure 7, the sender starts to broadcast at local time t_0. It continuously transmits preamble packets at t_1, t_2 and t_3. For the i^{th} preamble packet, the sender inserts the time interval $t_i - t_0$ into the preamble packet. On the receiver side, it detects the broadcast by channel sampling and it receives a preamble packet in a listen tail at local time t'_0. After a few software delay, the receiver successfully obtains the payload of the flooding packet at local time t'_1. The total delay is $t'_1 - t'_0$ at the receiver end. Since the SFD (Start Frame Delimiter) interrupt of a preamble packet happens at the same time for both sender and receiver, t_2 and t'_0 indicate the same time at

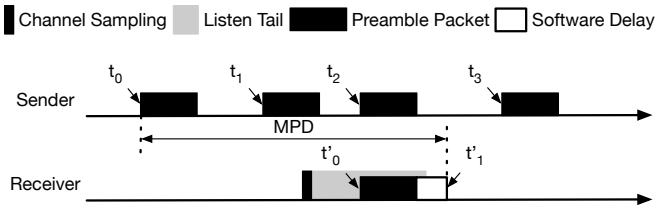


Fig. 7: The illustration of MPD measurement by using MAC layer timestamp.

Algorithm 4 Opportunistic Sender Selection Algorithm

Input: The ETD of sender ETD(s), ETD(i) and MPD(i) of node i.
Output: i is selected as sender or not.
1: **if** MPD(i) $\leq T_{sp}$ **then**
2: i is selected as sender with probability Max($1 - \frac{\text{MPD}(i)}{T_{sp}}$, 0).
3: **end if**
4: **if** ETD(i) − ETD(s) $> T_{ll}$ **then**
5: i is selected as sender and $W(i)$ is set as T.
6: **end if**

different clocks. Hence, the MDP of the receiver is calculated as $(t_2 - t_0) + (t'_1 - t'_0)$. In this way, node i obtains its MPD(i).

As shown in Algorithm 4, we illustrate our strategy of opportunistic sender selection which exploits both shortcut path and long link opportunities.

1) Shortcut Path: (line 1-3) For node i, if MPD(i) is smaller than a threshold T_{sp}, i has potential to serve as an opportunistic sender. However, since the risk of tail time increasing, it is not wise to select all of these potential nodes as senders. To prohibit the total number of opportunistic senders, i uses a probability based method to determine whether to transmit. The shorter MPD(i) is, the larger the probability is.

2) Long Link: (line 4-6) A sender attaches its ETD to its flooding packet. After node i receives a flooding packet, it compares ETD(i) with the ETD of the packet sender. If the difference is larger than a threshold T_{ll}, the flooding packet is received from an long link. Then, node i is selected as a sender.

Since the $MPD(i)$ is not larger than T, when T_{sp} is close to T, given the same $MPD(i)$, a node has high probability to serve as a sender. Moreover, the minimum ETD difference between a sender and its children is $T/2$. T_{ll} should not be smaller than $T/2$ to avoid all nodes are selected as senders. We empirically discuss the setting of T_{sp} and T_{ll} in Section IV-C and Section IV-D.

To conclude, COFlood follows two steps to select senders in a distributed manner. First, all nodes use beacons to construct an energy-efficient flooding tree. If a node have non-zero children, it will immediately broadcast the received flooding packet. The length of broadcast is determined by the worst quality among its children links. To further reduce the completion time of network flooding, other nodes will opportunistically serve as senders and broadcast for a whole sleep interval, when the opportunities of shortcut path and long link appear.

D. Failure Recovery

Due to the limited number of total broadcasts and the lossy links, it is possible a node fails to receive any flooding packet. To quickly discover and recover the failure, according to the spacial and temporal features [1] of the sampled signals in the tail, a node can detect the concurrent broadcast. Then, after detecting the concurrent broadcast, if the node does not successfully receive any flooding packet in a duration of two sleep interval, it will broadcast a request beacon to ask its parent node to rebroadcast the flooding packet until it successfully receives the flooding packet.

IV. Implementation

We implement COFlood with TinyOS 2.1.2 on TelosB motes. The RAM and ROM consumption are 3420 and 20806 bytes, respectively. Next, we discuss several practical issues during implementation.

A. Link Measurement and Filtering

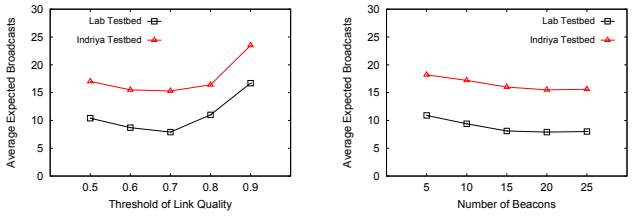
The link properties (i.e., PRR, neighbor relation) are the key information to construct flooding tree and determine the opportunistic sender. In our implementation, each node will broadcast 10 link beacon packets to measure the link quality in turn. For node i, it receives $\text{Rec}_i(j)$ beacon packets from node j. The PRR $p_i(j)$ of the link from node j to node i can be calculated as $p_i(j) = \text{Rec}_i(j)/10$. In this way, COFlood initializes the link quality between any two nodes.

Neighbor relation is another important information for flooding tree construction. COFlood sets a threshold p_n as the minimum quality of a neighbor link. Namely, a node treats another node as a neighbor only if the quality of the link between them is not less than p_n. If p_n is small, a node can cover more neighbors, but it needs to spend more broadcasts to guarantee the reliability. Hence, the total energy consumption might be high. Moreover, the larger the p_n is, the less the available links are. As a result, more senders are needed to guarantee the full connectivity. The total energy consumption might be also high.

We evaluate the energy efficiency of flooding tree when set different p_n from 0.5 to 0.9 on two testbeds. For each p_n, we run COFlood 10 times and calculate the average expected broadcasts of the flooding trees (i.e., E_b the sum of senders' expected broadcasts). The number of tree beacons is set as 20. In Figure 8(a), the results show that the average expected broadcasts is getting large when p_n is smaller or larger than 0.7 on both Lab testbed and Indriya testbed. This verifies the inference above. Hence, we set p_n as 0.7 to guarantee the energy efficiency of flooding tree.

B. Flooding Tree Initialization

After the link quality measurement, all nodes begin to initialize the flooding tree. Sink immediately broadcasts tree beacons and other nodes start to broadcast tree beacons when it has parent. We empirically set the number of tree beacons to tradeoff between the energy efficiency of flooding tree and initialization time. More tree beacons can guarantee the network consistency and make sure the flooding tree has the best energy efficiency. However, the less tree beacons are, the less initialization time is. Our goal is to find out the minimum number of beacons to make sure it converges to the most energy-efficient flooding tree.



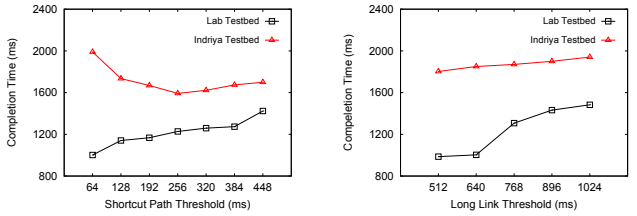
(a) Link Quality Threshold (b) Beacon Number

Fig. 8: The average expected broadcasts under different settings of link quality threshold (a) and beacon number (b) on Lab Testbed and Indriya Testbed.

We evaluate the energy efficiency of flooding tree given different number of tree beacons (e.g., 5, 10, 15, 20 and 25) on the two testbeds. For each setting, we run COFlood 10 times and measure the average expected broadcasts of the flooding trees. The p_n is set as 0.7. As it shown in Figure 8(b), the average expected broadcasts of the flooding trees decreases with the number of tree beacons increases. When the number of tree beacons is not less than 15, the energy efficiency of the flooding trees is stable on both Lab Testbed and Indriya Testbed. To keep the initialization time as low as possible, we set the number of tree beacons as 15 on the two testbeds.

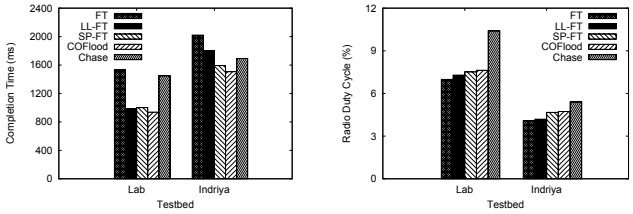
C. Shortcut Path Threshold

According to Algorithm 4, T_{sp} has great influence on the number of the senders who exploit shortcut path opportunity. If T_{sp} is too short, few of senders can be selected so that it fails to take the advantage of shortcut path. More senders are selected when T_{sp} is long. The tail time also increases so that enlarges the completion time. We disable long link opportunity, then evaluate the completion time of COFlood under 6 different T_{sp} from 64ms to 448ms. For each T_{sp}, we run network flooding 100 times to calculate the average completion time on the two testbeds. Other settings are same with Section V. As it shown in Figure 9(a), on Indriya Testbed, we achieve the minimum completion time when set T_{sp} as 256ms. On Lab Testbed, the completion time increases with the increasing of T_{sp}. The completion time is the smallest when T_{sp} is 64ms. The reason is that the node density of Lab Testbed is higher than Indriya Testbed (Section II). With high node density, a node has a large set of 1-hop neighbors. We can obtain the sufficient number of shortcut path senders with a small T_{sp}. In contrast, with small node density, to achieve the same number of shortcut path senders, we need to set a large T_{sp}. For COFlood evaluation on Lab Testbed and Indriya Testbed, we set T_{sp} as 64ms and 256ms.



(a) Shortcut Path Threshold (b) Long Link Threshold

Fig. 9: The completion time under different settings of shortcut path threshold (a) and long link threshold (b) on Lab Testbed and Indriya Testbed.



(a) Completion Time (b) Radio Duty Cycle

Fig. 10: The comparison of completion time (a) and radio duty cycle (b) among different protocols on Lab Testbed and Indriya Testbed.

D. Long Link Threshold

Similarly, T_{ll} has great influence on the number of long link senders. If T_{ll} is too large, the long link will be very lossy so that the number of long link senders is small. Moreover, T_{ll} should not be smaller than $T/2$. Otherwise, all nodes will be selected as sender and the overall performance is degraded. We disable shortcut path opportunity, then evaluate the completion time of COFlood under 5 different T_{ll} from 512ms to 1024ms. For each T_{ll}, we run network flooding 100 times to calculate the average completion time on the two testbeds. Other settings are the same with Section V. In Figure 9(b), we can see the completion time monotonically increases with the increasing of T_{ll} on both Lab Testbed and Indriya Testbed. The reason is that no long link is available when T_{ll} is getting large. Meanwhile, setting T_{ll} as 512ms will not make a large number of long link senders are selected so that keeps the tail time low. Hence, we set T_{ll} as 512ms for COFlood.

V. Evaluation

In this section, we show the performance of COFlood through the experiments on two real testbeds (i.e., Lab Testbed and Indriya Testbed) in terms of completion time and radio duty cycle. To evaluate the efficiency of each function component (i.e., flooding tree, shortcut path, long link), we choose four different combinations, namely FT (flooding tree), SP-FT (flooding tree + shortcut path), LL-FT (flooding tree + long link) and COFlood (flooding tree + shortcut path + long link). We further compare COFlood with Chase [1], which is state-of-the-art concurrent flooding protocol in asynchronous duty cycle networks. For all experiments, we set the sleep interval as 512ms. The packet length is 69 bytes. The channel is set as 26 and 22 for Lab Testbed and Indriya Testbed. The transmission power is set as 2 and 31, respectively. The IDs of the root nodes are 0 and 3. For each protocol, we run network flooding 100 times to measure the average completion time and radio duty cycle.

A. Completion Time

The results of completion time are shown in Figure 10(a). On Lab Testbed, the completion time of FT, LL-FT and SP-FT is 1530.2ms, 985.9ms and 1001.6ms. In comparison with FT, with long link and shortcut path, the completion time is shorten by 35.6% and 34.5%. COFlood combines the opportunities of long link and shortcut path. Its completion time is 937.7ms, which is 4.5% and 6.4% lower than LL-FT and SP-FT. We can see long link and shortcut path have similar contribution to shorten the completion time. The reason is the network is dense on Lab Testbed. We only need a small set of opportunistic senders to achieve the shortest completion time. Hence, both long link and shortcut path are very effective. Furthermore, in comparison with the completion time of *Chase* 1450ms, that of COFlood is 35.3% shorter.

On Indriya Testbed, the completion time of FT is 2020.8ms. By separately utilizing the opportunities of long link and shortcut path, the completion time of LL-FT and SP-FT is 1802.8ms and 1592.3ms, which is 10.8% and 21.2% shorter than FT. Moreover, the completion time of COFlood is 1496.8ms, which is 25.9% lower than FT. We can see shortcut path contributes more benefits than long link. The reason is on Indriya Testbed, the number of available long links is less than that of available shortcut paths. More shortcut path opportunities will provide more benefits. Furthermore, the completion time of COFlood is 10.9% lower than that of *Chase*.

In comparison Indriya Testbed, COFlood can achieve larger performance improvement on Lab Testbed. The reason is the network density of Indriya Testbed is lower than Lab Testbed. On Lab Testbed, an opportunistic sender can benefit a large set of nodes. However, the coverage benefit of a opportunistic sender is limited on Indriya Testbed.

B. Energy Efficiency

The results of radio duty cycle are shown in Figure 10(b). Since the number of senders is the smallest in FT, the radio duty cycle of FT (i.e., Lab 7.0% and Indriya 4.1%) is the lowest on the two testbeds. The radio duty cycle increases when we add more opportunistic senders by exploiting the opportunities of long link and shortcut path. On Lab Testbed, the radio duty cycle of LL-FT and SP-FT is 4.3% and 7.5% higher than FT. On Indriya Testbed, the radio duty cycle of LL-FT and SP-FT is 2.7% and 14.5% higher than FT. This also verifies that the number of opportunistic senders is similar on Lab Testbed and the number of shortcut path opportunities is larger than long link opportunity on Indriya Testbed. The

radio duty cycle of COFlood is 7.6% and 4.7%, which is 8.6% and 14.6% higher than FT, on Lab Testbed and Indriya Testbed. In comparison with FT, COFlood loses a small part of energy efficiency to greatly shorten the completion time. In comparison with *Chase*, the radio duty cycle of COFlood is 26.6% and 12.4% lower on Lab Testbed and Indriya Testbed.

VI. Related Work

In radio always-on networks, many protocols [5] [8] [9] [14] [16] [21] [22] of network flooding have been proposed. These protocols aim to optimize the delivery reliability and energy utilization with the constrains of channel contention/collision [9] [16], link properties [5] [21] [22] and network topology [8] [14]. However, in asynchronous duty cycle networks, the sleep delay and tail delay significantly enlarge the end-to-end delay and energy consumption. Hence, these protocols are hard to be directly adopted, but COFlood bridges the gap.

To shorten the unexpected sleep delay and tail delay in asynchronous duty cycle networks, some protocols [7] [15] [17] of network flooding have been well designed. However, S. Guo et al. [7] and F. Wang et al. [17] assume the active schedules of all neighbors are explicit for any node. In ADB [15], nodes send short probes to negotiate with their neighbors during network flooding. In comparison, COFlood does not rely on any requirement of time synchronization and continuous neighbor negotiation. The cost of network initialization is lower and the network reliability is higher, especially in large scale networks.

Recently, concurrent flooding [1] [6] is a promising way to speed up network flooding. Glossy [6] exploits the constructive interference to enable concurrent transmission in whole network. All nodes must keep awake simultaneously and relay packet with strict timing constraints. To further improve the reliability of constructive interference, several protocols [4] [19] exploits adaptive flooding structure to control the number of concurrent senders. Glossy based concurrent flooding does not work in asynchronous duty cycle networks, but COFlood works well.

Chase [1] exploits capture effect to enable concurrent broadcast in asynchronous duty cycle networks. Moreover, *Chase*++ [2] uses rateless coding to improve the performance of *Chase* when the packet payload becomes long. In parallel, based on *Chase*, COFlood further improves the energy consumption and completion time of concurrent flooding in asynchronous duty cycle networks.

VII. Conclusion

In this paper, we empirically study the state-of-the-art concurrent flooding protocol *Chase* on two real testbeds. We find that sender selection is a critical problem for concurrent flooding. Then, we propose COFlood, a practical and efficient sender selection method, to achieve both fast concurrent flooding and efficient energy consumption. First, COFlood constructs an energy-efficient flooding tree to cover the entire network with minimum energy cost. Moreover, to speed up concurrent flooding, COFlood exploits two potential opportunities, namely shortcut path and long link. We implement COFlood in TinyOS and evaluate its performance on two real testbeds. The results show that both completion time and radio duty cycle are reduced on the two testbeds.

References

[1] Z. Cao, D. Liu, J. Wang, and X. Zheng. Chase: Taming concurrent broadcast for flooding in asynchronous duty cycle networks. *IEEE/ACM Transactions on Networking*, 2017.

[2] Z. Cao, J. Wang, D. Liu, X. Miao, Q. Ma, and X. Mao. Chase++: Fountain-enabled fast flooding in asynchronous duty cycle networks. In *Proceedings of INFOCOM*, 2018.

[3] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda. Indriya: A low-cost, 3d wireless sensor network testbed. In *Proceedings of TRIDENTCOM*, 2011.

[4] M. Doddavenkatappa, M. C. Chan, and B. Leong. Splash: fast data dissemination with constructive interference in wireless sensor networks. In *Proceedings of NSDI*, 2013.

[5] W. Dong, Y. Liu, C. Wang, X. Liu, C. Chen, and J. Bu. Link quality aware code dissemination in wireless sensor networks. In *Proceedings of ICNP*, 2011.

[6] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of IPSN*, 2011.

[7] S. Guo, L. He, Y. Gu, B. Jiang, and T. He. Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links. *IEEE Transactions on Computers*, 63(11):2787–2802, 2014.

[8] L. Huang and S. Setia. Cord: Energy-efficient reliable bulk data dissemination in sensor networks. In *Proceedings of INFOCOM*, 2008.

[9] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of Sensys*, 2004.

[10] O. Landsiedel, E. Ghadimi, S. Duquennoy, and M. Johansson. Low power, low delay: opportunistic routing meets duty cycling. In *Proceedings of IPSN*, 2012.

[11] X. Mao, X. Miao, Y. He, X.-Y. Li, and Y. Liu. Citysee: Urban co 2 monitoring with sensors. In *Proceedings of INFOCOM*, 2012.

[12] L. Mo, Y. He, Y. Liu, J. Zhao, S.-J. Tang, X.-Y. Li, and G. Dai. Canopy closure estimates with greenorbs: sustainable sensing in the forest. In *Proceedings of Sensys*, 2009.

[13] D. Moss and P. Levis. Box-macs: Exploiting physical and link layer boundaries in low-power networking. *Technical Report SING-08-00, Stanford*, 2008.

[14] V. Naik, A. Arora, P. Sinha, and H. Zhang. Sprinkler: A reliable and energy efficient data dissemination service for extreme scale wireless networks of embedded devices. *IEEE Transactions on Mobile Computing*, 6(7):777–789, 2007.

[15] Y. Sun, O. Gurewitz, S. Du, L. Tang, and D. B. Johnson. Adb: an efficient multihop broadcast protocol based on asynchronous duty-cycling in wireless sensor networks. In *Proceedings of Sensys*, 2009.

[16] G. Tolle and D. E. Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proceedings of EWSN*, 2005.

[17] F. Wang and J. Liu. On reliable broadcast in low duty-cycle wireless sensor networks. *IEEE Transactions on Mobile Computing*, 11(5):767–779, 2012.

[18] J. Wang, Z. Cao, X. Mao, and Y. Liu. Sleep in the dins: Insomnia therapy for duty-cycled sensor networks. In *Proceedings of INFOCOM*, 2014.

[19] Y. Wang, Y. He, X. Mao, Y. Liu, and X.-Y. Li. Exploiting constructive interference for scalable flooding in wireless networks. *IEEE/ACM Transactions on Networking*, 21(6):1880–1889, 2013.

[20] X. Zheng, Z. Cao, J. Wang, Y. He, and Y. Liu. Zisense: towards interference resilient duty cycling in wireless sensor networks. In *Proceedings of Sensys*, 2014.

[21] X. Zheng, J. Wang, W. Dong, Y. He, and Y. Liu. Bulk data dissemination in wireless sensor networks: analysis, implications and improvement. *IEEE Transactions on Computers*, 65(5):1428–1439, 2016.

[22] T. Zhu, Z. Zhong, T. He, and Z.-L. Zhang. Exploring link correlation for efficient flooding in wireless sensor networks. In *Proceedings of NSDI*, 2010.